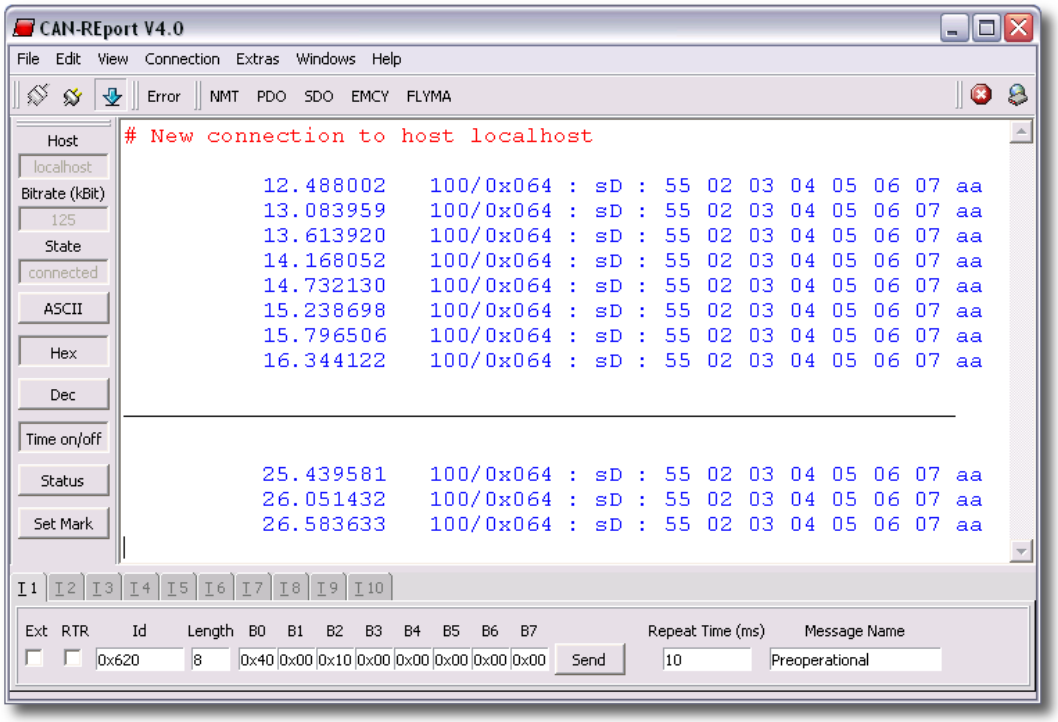
CAN-RE*port*
User Manual



© *port* GmbH, Halle 2010; CAN-REport Version 4.0.3

Disclaimer
All rights reserved

The programs, boards and documentations supplied by *port* GmbH are created with due diligence, checked carefully and tested on several applications.

Nevertheless, *port* GmbH can not take over no guarantee and no assume del credere liability that the program, the hardware board and the documentation are error-free respective are suitable to serve the special purpose.

In particular performance characteristics and technical data given in this document may not be constituted to be guaranteed product features in any legal sense.

For consequential damages, which are emerged on the strength of use the program and the hardware boards therefore, every legal responsibility or liability is excluded.

port has the right to modify the products described or their documentation at any time without prior warning, as long as these changes are made for reasons of reliability or technical improvement.

All rights of this documentation lie with *port*. The transfer of rights to third parties or duplication of this document in any form, whole or in part, is subject to written approval by *port*. Copies of this document may however be made exclusively for the use of the user and his engineers. The user is thereby responsible that third parties do not obtain access to these copies.

The soft- and hardware designations used are mostly registered and are subject to copyright.

We are thankful for hints of possible errors and may ask around for an information.

We will go all the way to verify such hints fastest

Copyright

© 2010 *port* GmbH
Regensburger Straße 7b
D-06132 Halle
Tel. +49 345 - 777 55 0
Fax. +49 345 - 777 55 20
E-Mail service@port.de
Internet http://www.port.de

Table of Contents

1. Overview 5

2. Installation 6

3. Application 6

4. Interfacing with CAN 7
4.1. TCP/IP 7
4.1.1. CAN-Server 7
4.1.2. Client - *CAN-REport* 7
4.2. Serial Interfaces 8

5. Graphical User Interface 9
5.1. Menubar 9
5.1.1. File Menu 9
5.1.2. Edit Menu 10
5.1.3. View Menu 10
5.1.3.1. View mode 11
5.1.4. Connection Menu 11
5.1.5. Extras Menu 13
5.1.5.1. Options 15
5.1.6. Window Menu 16
5.1.7. Help Menu 16
5.2. Toolbar 17
5.3. Statusbar 17
5.4. Transmit Toolbar 18
5.5. Statistical Tooltip 19
5.6. CAN Log window 19

6. Interpretation of CAN messages 21

7. CANopen-Plugin 23
7.1. Set Node Names 23
7.2. Service oriented interpretation 23

7.2.1. PDO Configuration 23
7.2.1.1. DCF-Import 24

8. DeviceNet-Plugin 26

9. J1939-Plugin 26

10. RS232-Plugin 28

11. Trigger 30
11.1. Defining a trigger CAN message 31
11.2. Defining a trigger Remote Request CAN message 31
11.3. CANopen Message Builder 31

12. Filter 33
12.1. Hardware Filter 33
12.2. Software Filter 34

13. Programming with *CAN-REport* 35
13.1. Extending *CAN-REport* 35
13.2. Interpretation of CAN messages 39
13.2.1. Example 44
13.3. Multichannel functions 45
13.4. Testscript commands 45
13.5. CANopen functions 45
13.6. DeviceNet functions 51
13.7. J1939 functions 59
13.8. RS232 functions 60
13.9. Extending the user interface 61

14. Configuration 63
14.1. Serial interfaces 63
14.1.1. CAN232 - Configuration 63
14.1.2. CANview - Configuration 63

15. System Requirements 64

1. Overview

The CAN-Analyzer *CAN-REport* from *port* is an efficient and versatile usable instrument for development, test and maintenance, for analysis and commissioning of CAN-based networks like DeviceNet and CANopen.

The built-in scripting capability allows a universal usage for development and test of CAN devices besides the normal possibilities of displaying the received CAN messages. It is especially useful in the field of industrial CAN networking.

The separation of hardware-interface (CAN access) and visualization software allows the usage in TCP/IP networks.

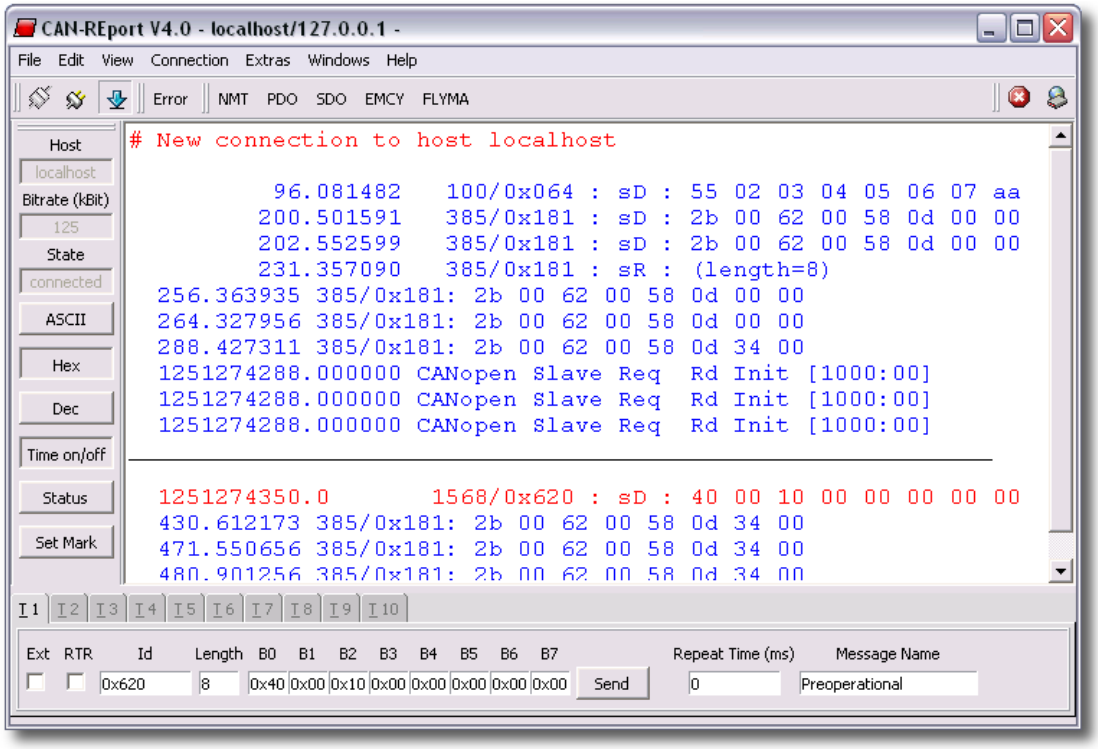


Figure 1, Analyzer view

Note: The performance of the *CAN-REport* depends on the used CAN interface hardware. Especially at high bus load and high baud rates some CAN messages may be lost.

2. Installation

The installation includes:

- the graphical user interface
- the horch server and
- a layer 2 driver for the CAN interface.

For the installation execute setup.exe.

The installation of all software components is done automatically and menu driven. This includes copying of all manuals.

3. Application

In the standard equipment the CAN-Analyzer already has a lot of efficient basic functions at its disposal, which cover most application needs.

Among other things the on-line observation of the bus traffic, the transmitting of unique or cyclical CAN messages and whole message sequences as well as the recording of CAN messages and the storage to log files belong to this purpose. Besides the standard text format, we also support the CSV format for easy data transfer into spread sheet applications.

Application specific interpretation of the messages' data content can be defined through a project specific data base.

Through supplementary software modules extended functionality can be made available. These are among other things the protocol-specific representations in separate protocol windows for different higher layer protocol messages. Figure 2 shows this for SDO messages in CANopen-based networks.

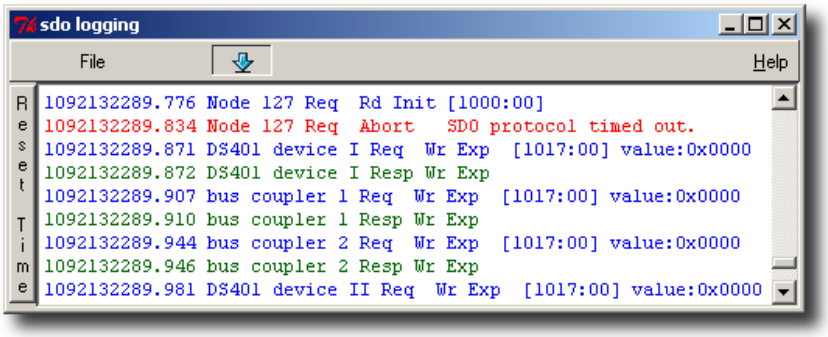

sdo logging
File
Help
Reset Time
1092132289.776 Node 127 Req Rd Init [1000:00]
1092132289.834 Node 127 Req Abort SDO protocol timed out.
1092132289.871 DS401 device I Req Wr Exp [1017:00] value:0x0000
1092132289.872 DS401 device I Resp Wr Exp
1092132289.907 bus coupler 1 Req Wr Exp [1017:00] value:0x0000
1092132289.910 bus coupler 1 Resp Wr Exp
1092132289.944 bus coupler 2 Req Wr Exp [1017:00] value:0x0000
1092132289.946 bus coupler 2 Resp Wr Exp
1092132289.981 DS401 device II Req Wr Exp [1017:00] value:0x0000

Figure 2, CANopen SDO interpretation and recording

You can see the symbolic representation of service and node name, "bus coupler 1 Req", as well as displaying of "index:subindex" within the **SDO Initiate** requests.

With the help of the built-in scripting language programming of new functions or individual plugins with a graphic interface can be programmed easily.

4. Interfacing with CAN

4.1. TCP/IP

The analyzer *CAN-REport* consists of the CAN hardware interface and the visualization software. Both are connected as server and client by a standard TCP/IP network connection. This separation allows the usage of the CAN interface as a remote interface. That means both parts can be located at different computer systems. Remote monitoring of CAN networks is possible without additional or modified software over LAN, dial or internet connections.

The **Horch**-protocol is used as a communication protocol between operating-client and hardware-server on top of TCP/IP.

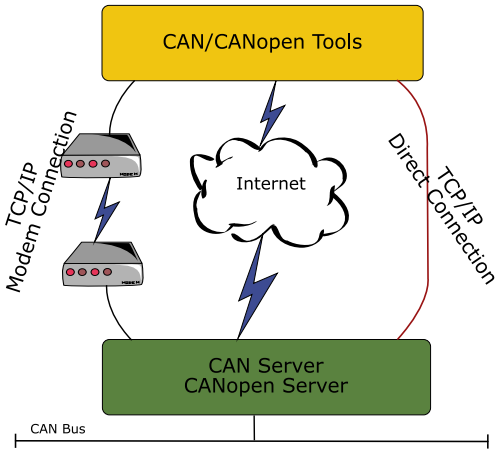


Figure 3, Client-Server communication

CAN interfaces are currently available for ISA, PCI, parallel port, serial, and PC104 with MS-Windows operating systems, ISA and PCI for LINUX or as Ethernet-interface as stand alone CAN-Server on the *port*-EtherCAN module.

4.1.1. CAN-Server

The CAN-Server *horch* is the interface from TCP/IP to CAN. A detailed description can be found in its manual.

4.1.2. Client - *CAN-REport*

The visualization and operating software is based on the scripting language *Tcl/Tk* which is available for many different computer platforms. The usage is therefore possible on all systems supporting this language. Through separation of hardware CAN-Interface and man-machine interface it is possible to use the *CAN-REport* besides MS-Windows on LINUX computers for controlling the CAN hardware interface.

4.2. Serial Interfaces

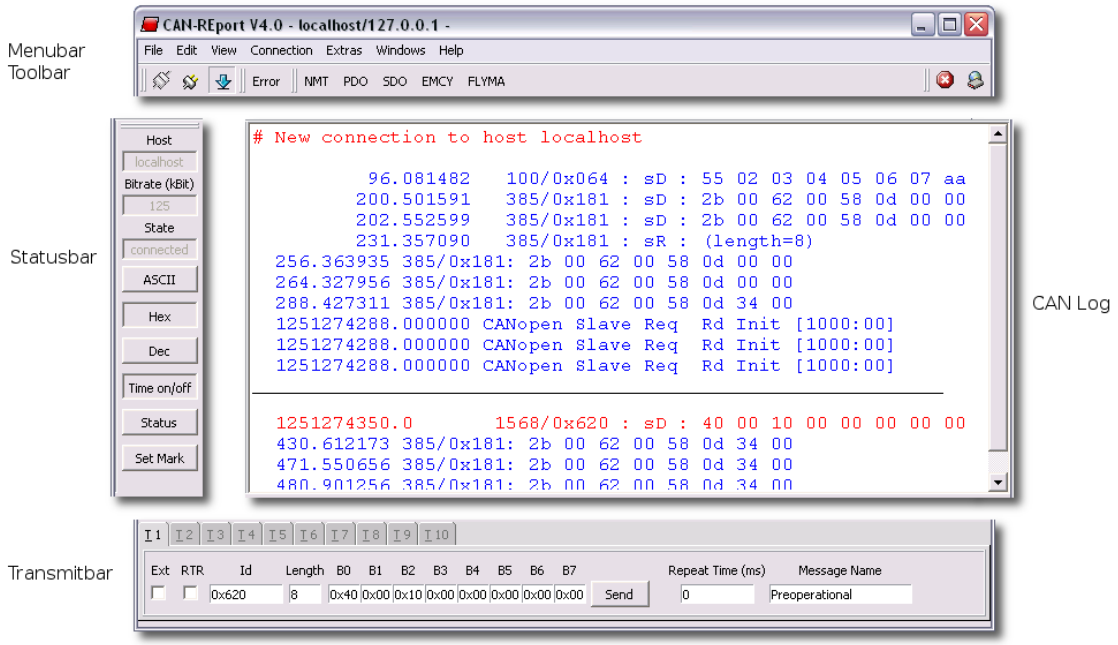
Besides the client-server connections the *CAN-REport* supports also direct connections to CAN-RS232 interfaces as:

- CAN232, CANUSB *(Lawicel)*
- CANview *(RM Michaelides)*

For further information about the configuration please have a look at the section Configuration of serial interfaces.

5. Graphical User Interface

The graphical user interface consists of 4 components.



5.1. Menubar

5.1.1. File Menu

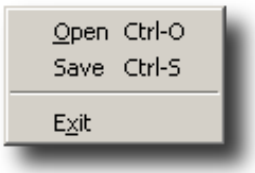


Figure 4, File Menu

Open
: Open a previously stored logfile. If the *CANopen Plugin* is activated then the contents can be interpreted as CANopen data.

Save
: Save the contents of the main window into a file.

Exit
: Exits *CAN-REport*.

5.1.2. Edit Menu

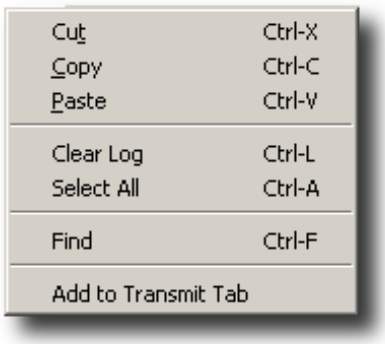


Figure 5, Edit Menu

Cut
Cut selected text into clipboard.

Copy
Copy selected text into clipboard.

Paste
Pastes text from clipboard into the main window.

Clear Log
Clears content of the main logging window.

Select All
Selects the whole content of the main logging window.

Find
Opens a standard find dialog box to search for text signatures.

Add to Transmit Tab
Copies the CAN message from the cursor to the currently active transmit tab.

5.1.3. View Menu

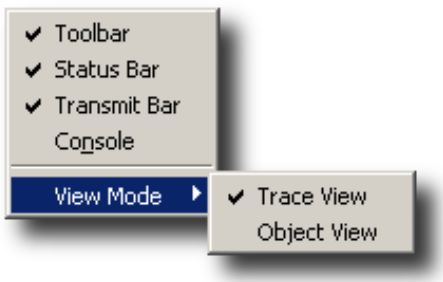


Figure 6, View Menu

Toolbar
Toggles the view of the toolbar.

Statusbar
Toggles the view of the statusbar.

Transmitbar
Toggles the view of the statusbar.

Console
Opens the Tcl-Tk console as a new window. With the console you can extend the functionality of *CAN-REport.* See paragraph **Extending** *CAN-REport.*

5.1.3.1. View mode

CAN-REport possesses a trace mode and a object view mode.

Trace view
All CAN messages are written into the main window as they are received. The view shows: timestamp, CAN-ID, type, data.

Object view
Each CAN message is placed in one line. Every time the message with the same CAN identifier is received the line is updated. A maximum of 192 messages can be displayed. The view shows: CAN-ID, type, data, period and a receive counter.

5.1.4. Connection Menu

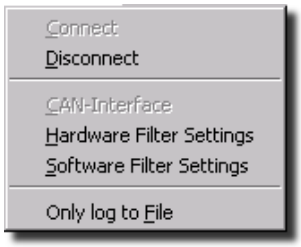


Figure 7, Connection Menu

Connect
Connect to the CAN-Server *horch*. All necessary data, i.e. server name, port and bitrate, are input within the menu *CAN-Interface*.

Disconnect
Disconnect from the server.

CAN-Interface
Opens a dialog for the configuration of the TCP/IP connection to the CAN-Server. Only the interfaces that are installed with an appropriate driver are displayed. For serial interfaces no separate driver is needed.

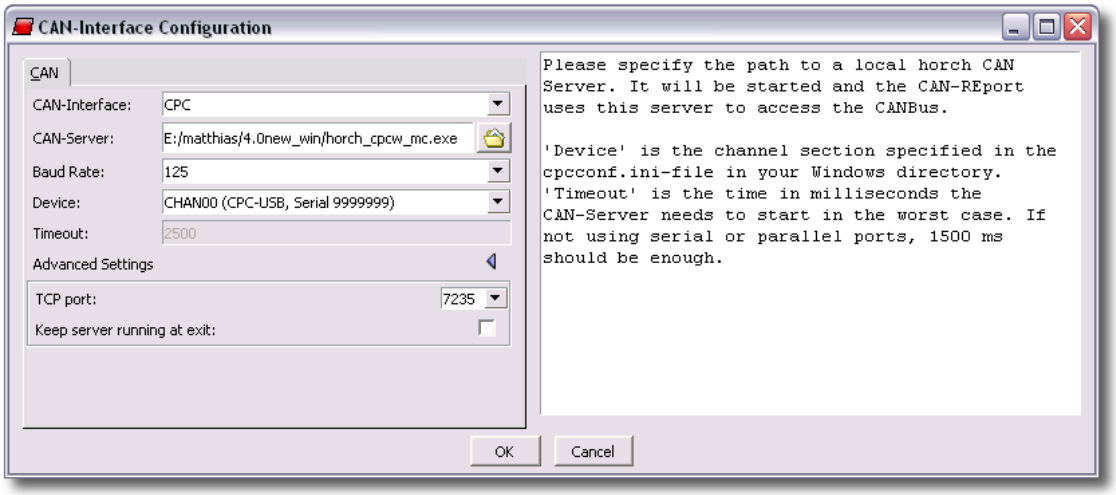

CAN-Interface Configuration
CAN
CAN-Interface:
CPC
CAN-Server:
E:/matthias/4.0new_win/horch_cpcw_mc.exe
Baud Rate:
125
Device:
CHAN00 (CPC-USB, Serial 9999999)
Timeout:
2500
Advanced Settings
TCP port:
7235
Keep server running at exit:
Please specify the path to a local horch CAN Server. It will be started and the CAN-REport uses this server to access the CANBus.
'Device' is the channel section specified in the cpcconf.ini-file in your Windows directory. 'Timeout' is the time in milliseconds the CAN-Server needs to start in the worst case. If not using serial or parallel ports, 1500 ms should be enough.
OK
Cancel

Figure 8, Configuration dialog of CAN interface

Hardware Filter Settings

Opens a dialog for setting the acceptance mask register of the CAN controller. Although this menu is provided not every hardware interface supports the setting of registers. See paragraph Filter.

Software Filter Settings

Opens a dialog for setting CAN ids that are to be received. Filtering is carried out by the CAN server horch. Interfaces that do not use the horch server do not support this feature. See paragraph Filter.

Trigger

Opens a window for trigger configuration (see paragraph Trigger).

Only Log to File

CAN messages are saved to a log file and are not displayed in the main window. The data can be logged in one single file or in multiple files. For logging in multiple files a time interval can be specified when a new file will be started. The filenames are placed in a directory that has the format: *Year-Month-Day.* The file itself has the format: *YearMonthDay_HourMinute.log.*

Logging in files supports two formats: plain text files and CSV files.

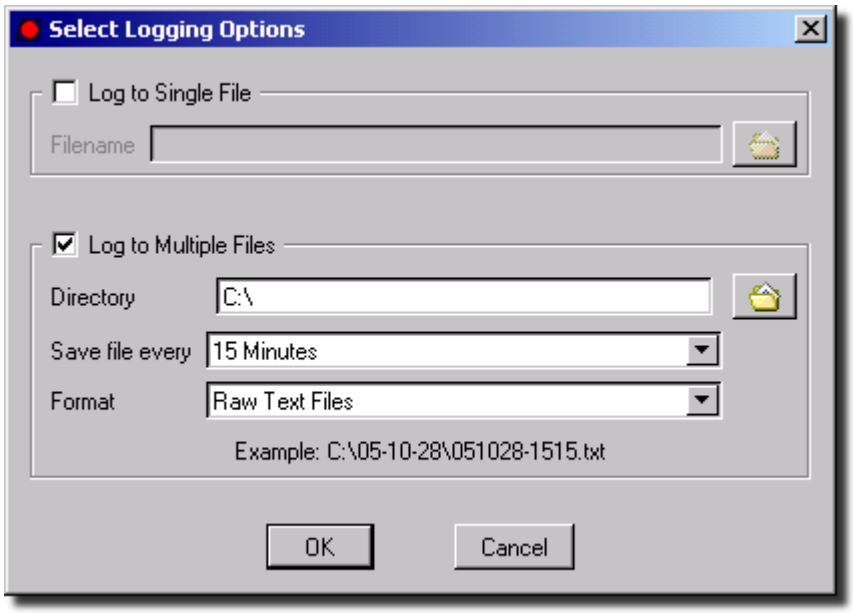


Figure 9, Log file dialog

5.1.5. Extras Menu

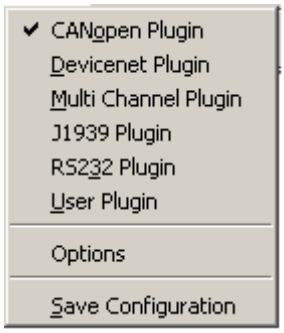


Figure 10, Extras Menu

CANopen Plugin

Loads the CANopen plugin. If switched on the toolbar is extended by buttons for the CANopen services SDO, PDO, NMT, EMCY, FLYMA. When clicking on a button a windows pops up and displays the CAN message of its service.

DeviceNet Plugin

Loads the DeviceNet plugin. If switched on the toolbar is extended with a button for the DeviceNet log window. When clicking on a button a window pops up and displays a descriptive text for the message.

Multichannel Plugin

Loads the multichannel Plugin. It is used for monitoring 2 CAN lines at a time. Therefore it needs to connect to two CAN-servers.

When the plugin is activated when a connection is open it closes the current connection. With the multichannel plugin the CAN servers need to be started manually and the CAN-Interface has to be set to **TCP**! The TCP port of the second CAN server has to be set to the port number of the first CAN server plus 1, ie. CAN server 1 uses port number 7235, CAN server 2 uses port number 7236.

With the USBCAN2 from SYS TEC the module is accessed with out TCP. CAN-REport uses instance 0 for CAN line 1 and instance 1 for CAN line 2.

The main window displays the CAN messages from both lines. The transmitbar is changed in that the checkbuttons for extended messages and RTR messages now are used for selecting the line. Further it provides script commands to send messages on both CAN lines.

J1939 Plugin

Loads the J1939 Plugin. The plugin uses databases for the interpretation form file *j1939def.tcl*. The database is defined as Tcl list that can be extended if needed. See paragraph J1939-Plugin.

RS232 Plugin

The RS232 Plugin allows to monitor a serial interface (see chapter RS232 Plugin).

User Plugin

Loads the file <application.tcl> and adds a button to the toolbar. This file provides a Tcl-function for user-defined protocol interpretation. An example file is provided.

5.1.5.1. Options

Opens a dialog to configure fonts, settings for Start and Exit and plugin specific settings like mapping for PDO, EMCY for the CANopen Plugin.

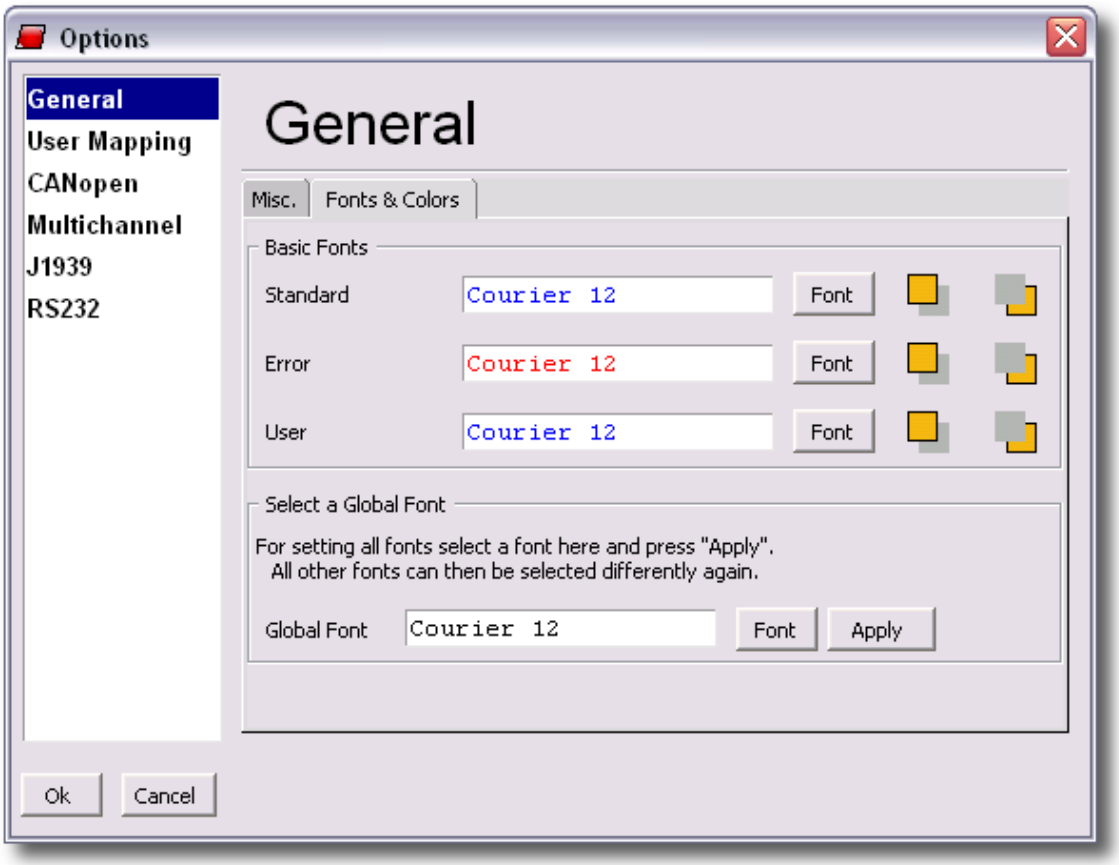


Figure 11, Options dialog

For User, PDO and EMCY mapping please see paragraph Interpretation of CAN messages.

CANopen interpretation in main window
: Activates CANopen interpretation in the main window.

Connect at Startup
: This option controls the startup behavior. When activated *CAN-REport* will try to connect to the CAN server at the next start.

Tooltips
: Activates tooltips for every button and input field.

Auto Save
: If this menu is activated then the configuration is stored before exiting *CAN-REport*.

5.1.6. Window Menu

Figure 12, Window Menu

Clear All
Clear the content of the main window and of windows that are created of plugins.

Cascade
Place the plugin windows cascaded next to the main window.

Tile Vertical
Place the plugin windows vertical next to the main window.

Tile Horizontal
Place the plugin windows horizontal under the main window.

Hide All
Hide all plugin windows.

Show All
Show all plugin windows.

5.1.7. Help Menu

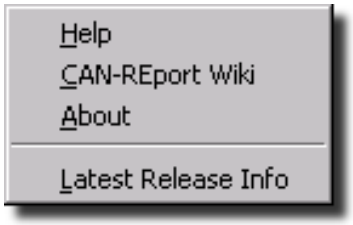


Figure 13, Help Menu

Help
Provides a short help to *CAN-REport*.

CAN-REport Wiki
Opens your standard browser and links to the Wiki pages of *CAN-REport*.

About

Copyright and license information. Additionally it displays the Tcl/Tk packages that are available.

Latest Release Info

Checks for the latest release of *CAN-REport*. Therefore it establishes a TCP/IP connection to *http://www.port.de*.

5.2. Toolbar

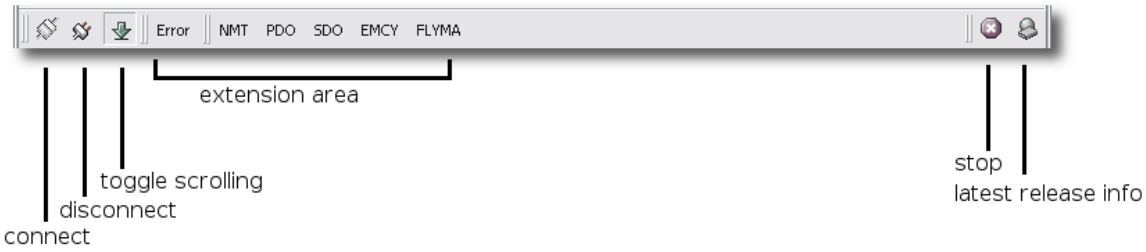


Figure 14, Toolbar

The toolbar provides quick access to the menu items Connect and Disconnect. The third button with the down arrow switches scrolling of the main window on or off. The stop button is used for canceling user-defined scripts. Please see paragraph: Extending *CAN-REport*. With the "www"-button *CAN-REport* checks if a new version of *CAN-REport* is available. Therefore it connects to the internet.

5.3. Statusbar

Figure 15, Statusbar

The Status shows information about the connection to the *horch*-Server. Additionally there are buttons influence the display of the CAN messages in the main window. The buttons "ASCII", "Hex" and "Dec" change the display of the data of a CAN message. With the "toggle Time"-button the display of the receiving time can be switched on and

off. It is displayed in "seconds.<parts of seconds>". The <parts of seconds> depends on the driver. The Linux driver *can4linux* supports a resolution of mircoseconds. With the button "Set Mark" a separation line is inserted in the main window.
The status button provides a window with information about the underlying CAN hardware and driver respectively. The figure 16 below shows the information of a CAN server that is running on Linux. Windows driver do not support all shown information. Therefore some values will always show 0.

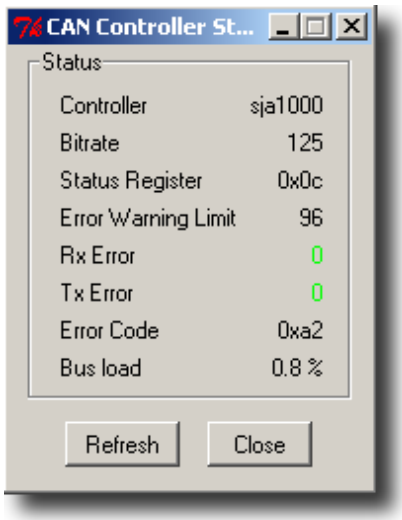


Figure 16, Status window

5.4. Transmit Toolbar

CAN-REport also allows it to send CAN messages. In the transmit toolbar at the bottom a complete CAN message can be specified with CAN id and CAN data. With the button "Send" the message is sent. The number of tabs can be set in the configuration file <can-report.rc> with:

```
set ST(txch) 12 ; #put any number here
```

The CAN messages are stored in the configuration file in the current working directory. If "AutoSave" was activated *CAN-REport* will save the settings automatically at exit. At the next start all your settings will be reloaded. Optionally a name can be assigned to a CAN message. The edit fields take decimal, hexadecimal and octal values as input. Hexadecimal values are prepended with "0x" and octal values with "0".

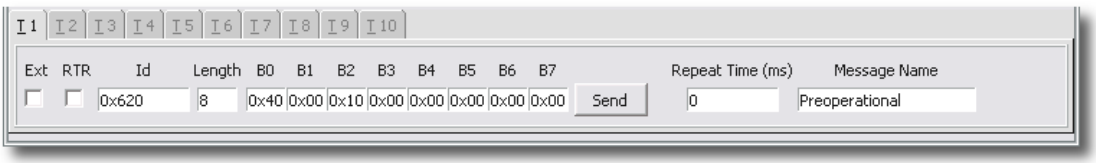

I1 I2 I3 I4 I5 I6 I7 I8 I9 I10
Ext RTR Id Length B0 B1 B2 B3 B4 B5 B6 B7 Repeat Time (ms) Message Name
0x620 8 0x40 0x00 0x10 0x00 0x00 0x00 0x00 0x00 Send 0 Preoperational

Figure 17, Transmit Toolbar

If it is necessary to send CAN messages periodically, then a "Repeat Time" in ms can be specified. The CAN message sent is not displayed in the main window. Only a notification is shown. If a "Repeat Time" was specified no notification is displayed.

One can assign a name to a message you want to send/save. You can do this by entering the name into the input field named "Message Name". **Hint**: if you are ready to enter the name than confim it by hitting the <ENTER> key and the given name will appear as title of the current selected transmitbar tab.

5.5. Statistical Tooltip

A small and basic statistical overview is shown in a tooltip when uninterpreted CAN messages in the main window are selected with the mouse pointer. The overview is shown in a tooltip when the left mouse button is released. While the left mouse button is pressed the keys «Ctrl + Pos1» and «Ctrl + End» jump to the top or bottom of the message log.

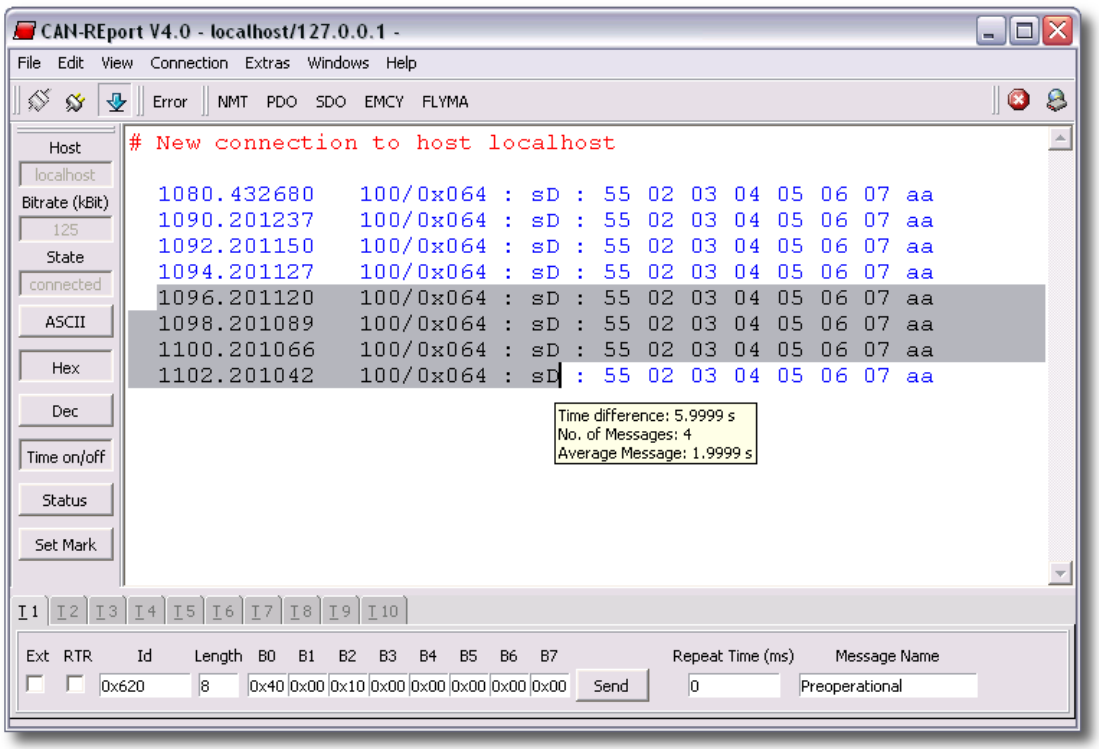

CAN-REport V4.0 - localhost/127.0.0.1 -
File Edit View Connection Extras Windows Help
Error NMT PDO SDO EMCY FLYMA
Host
localhost
Bitrate (kBit)
125
State
connected
ASCII
Hex
Dec
Time on/off
Status
Set Mark
New connection to host localhost
1080.432680 100/0x064 : sD : 55 02 03 04 05 06 07 aa
1090.201237 100/0x064 : sD : 55 02 03 04 05 06 07 aa
1092.201150 100/0x064 : sD : 55 02 03 04 05 06 07 aa
1094.201127 100/0x064 : sD : 55 02 03 04 05 06 07 aa
1096.201120 100/0x064 : sD : 55 02 03 04 05 06 07 aa
1098.201089 100/0x064 : sD : 55 02 03 04 05 06 07 aa
1100.201066 100/0x064 : sD : 55 02 03 04 05 06 07 aa
1102.201042 100/0x064 : sD : 55 02 03 04 05 06 07 aa
Time difference: 5.9999 s
No. of Messages: 4
Average Message: 1.9999 s
I1 I2 I3 I4 I5 I6 I7 I8 I9 I10
Ext RTR Id Length B0 B1 B2 B3 B4 B5 B6 B7
0x620 8 0x40 0x00 0x10 0x00 0x00 0x00 0x00 0x00
Send
Repeat Time (ms)
0
Message Name
Preoperational

Figure 18, Statistic tooltip

5.6. CAN Log window

The CAN logger displays CAN messages. It has all functions of a text editor. This provides the functionality to mark relevant blocks of CAN messages or add annotations.

CAN messages are show in the following format:

```
      11.123  256/0x100 : sD : 00 01 02 03 04 05 06 07
#     ^       ^           ^    ^
#       time  id dec/hex type data 1..8 (if available)
```

Line Element	Description
Time stamp	Time of reception in format: *seconds.milliseconds.* The accuracy/resolution is dependent of the operating system
CAN Id	Identifier of the CAN message as decimal and hexadecimal value
Type	sD - standard data frame (11 bit identifier) xD - extended data frame (29 bit identifier) sR - standard remote frame (11 bit identifier) xR - extended remote frame (29 bit identifier)
Data	Data bytes in the selected format. Hexadecimal is the standard. The prefix *0x* for the hexadecimal values is omitted for better readability

6. Interpretation of CAN messages

This dialog is used for User-, PDO and EMCY-Mapping. PDO and EMCY mapping is only available if the *CANopen Plugin* is activated. PDO/EMCY-Mapping is used to interpret CAN-messages with ID of the PDO (0x181 - 0x57F) and EMCY (0x81 - 0xFF) range in a user specific way. The user mapping can be used for all other CAN messages.

The data contents can be interpreted as signed and unsigned integer values. Each data item can be assigned to a user specific text and the data format ("big endian", "little endian") can be selected. Configuration is done with two dialogs. First there is an overview dialog that shows all configured messages. Second there is the message configuration dialog.

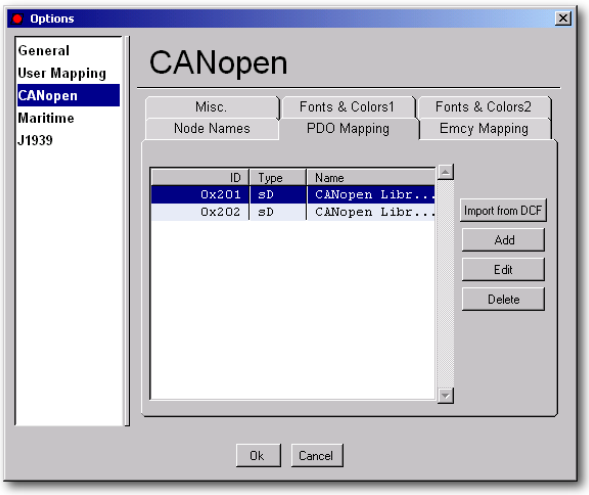


Figure 19, PDO/EMCY-Mapping dialog

The dialog has buttons to add, edit and delete messages you want to watch and interpret. On press of the add and edit button the message configuration dialog opens. The CANopen PDO mapping provides the functionality of importing a mapping of a *Device Configuration File* (DCF). See paragraph: CANopen Plugin.

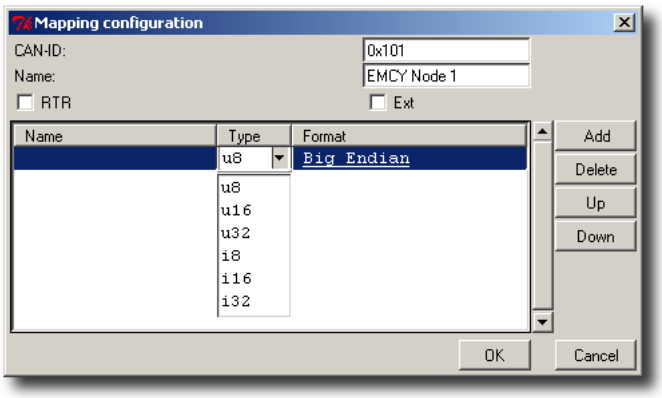


Figure 20, Message configuration dialog

Note: For EMCY messages the first 3 bytes of the CAN message are predefined as u16 and u8. This is not displayed and cannot be altered.

7. CANopen-Plugin

7.1. Set Node Names

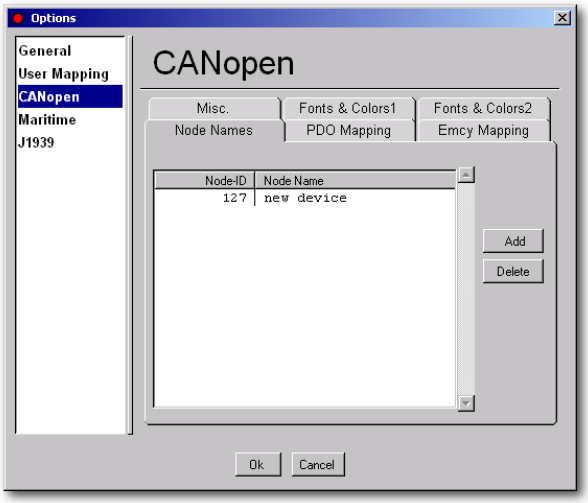


Figure 21, CANopen configuration of CANopen node names

This dialog is only available if *CANopen Plugin* is activated. With this dialog you can define a symbolic name for a CANopen device. This name is used for all CANopen service windows instead of the node id. Figure 22 shows an example.

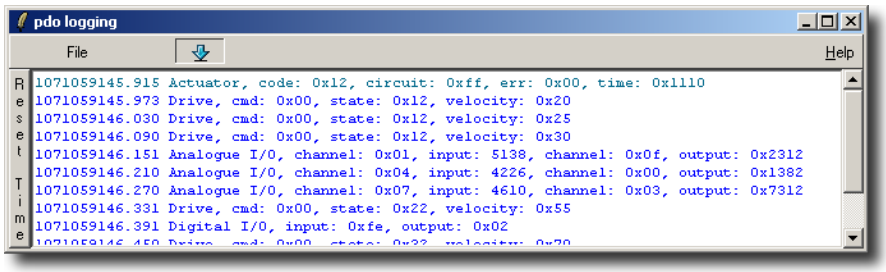

pdo logging
File
Help
Reset Time
1071059145.915 Actuator, code: 0x12, circuit: 0xff, err: 0x00, time: 0x1110
1071059145.973 Drive, cmd: 0x00, state: 0x12, velocity: 0x20
1071059146.030 Drive, cmd: 0x00, state: 0x12, velocity: 0x25
1071059146.090 Drive, cmd: 0x00, state: 0x12, velocity: 0x30
1071059146.151 Analogue I/O, channel: 0x01, input: 5138, channel: 0x0f, output: 0x2312
1071059146.210 Analogue I/O, channel: 0x04, input: 4226, channel: 0x00, output: 0x1382
1071059146.270 Analogue I/O, channel: 0x07, input: 4610, channel: 0x03, output: 0x7312
1071059146.331 Drive, cmd: 0x00, state: 0x22, velocity: 0x55
1071059146.391 Digital I/O, input: 0xfe, output: 0x02

Figure 22, CANopen PDO interpretation and recording

7.2. Service oriented interpretation

The CANopen plugin provides separate logging windows for the services: NMT, SDO, PDO, EMCY and Flying Master (FLYMA).
The interpretation for PDO and EMCY can be tailored alike the user mapping for the specific application.

7.2.1. PDO Configuration

Incomming PDO messages can be configured to be displayed with symbolic information. The databytes of the PDO are grouped to valid CANopen datatypes. Each datatype can be assigned to a symbolic text.

Whenever a PDO message is received the data is formatted as specified and displayed in the PDO window. In cases where there are many PDO messages on the network it is possible to switch on a filter that only shows the configured PDO messages. If the CANopen interpretation is shown in the main window then the PDO interpretation will be shown there, too.

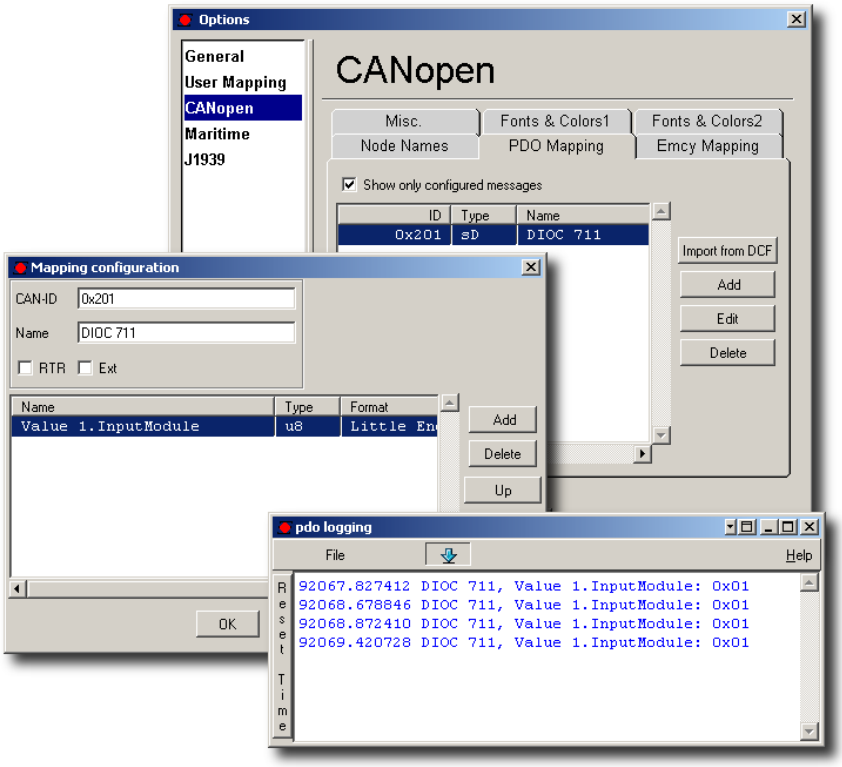


Figure 23, PDO Configuration with example output

7.2.1.1. DCF-Import

PDOs in a CANopen network are described by a set of DCF (Device Configuration Files). The DCF import dialog shows a tabular interface for assigning a CANopen node id to a DCF. The DCF can be selected by pressing the right mouse button. If no DCF file is available, EDS files can be imported. Expressions like are evaluated according to the node id.

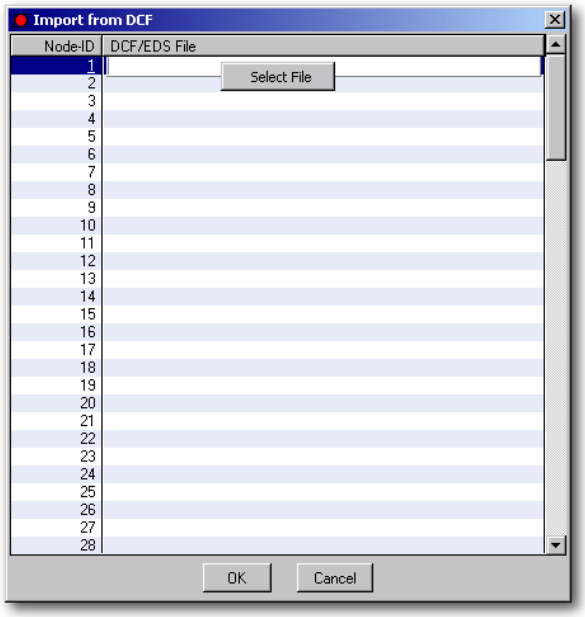


Figure 24, DCF import dialog

8. DeviceNet-Plugin

The DeviceNet plugin shows CAN-messages according to the DeviceNet protocol. It also provides scripting commands that can be used for writing test procedures. By using these functions reading and writing of object attributes is possible. See paragraph Extending *CAN-REport*.

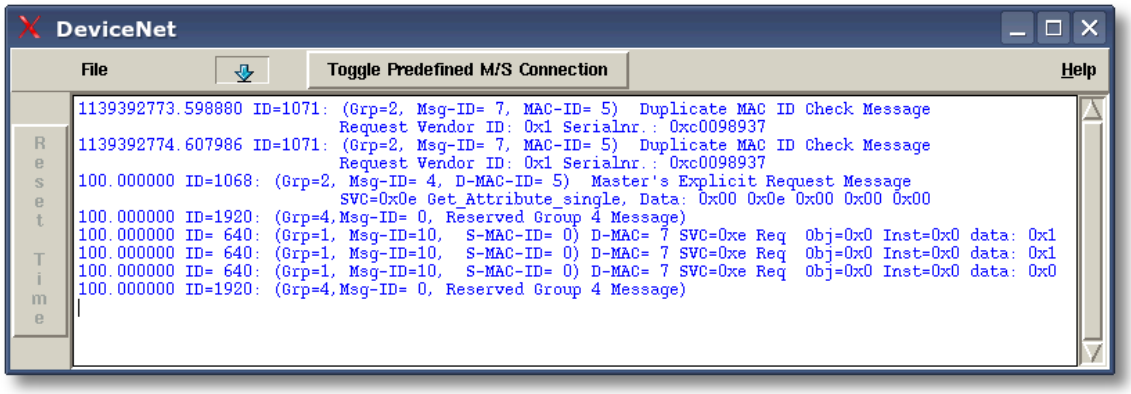

DeviceNet
File
Toggle Predefined M/S Connection
Help
Reset Time
1139392773.598880 ID=1071: (Grp=2, Msg-ID= 7, MAC-ID= 5) Duplicate MAC ID Check Message
Request Vendor ID: 0x1 Serialnr.: 0xc0098937
1139392774.607986 ID=1071: (Grp=2, Msg-ID= 7, MAC-ID= 5) Duplicate MAC ID Check Message
Request Vendor ID: 0x1 Serialnr.: 0xc0098937
100.000000 ID=1068: (Grp=2, Msg-ID= 4, D-MAC-ID= 5) Master's Explicit Request Message
SVC=0x0e Get_Attribute_single, Data: 0x00 0x0e 0x00 0x00 0x00
100.000000 ID=1920: (Grp=4,Msg-ID= 0, Reserved Group 4 Message)
100.000000 ID= 640: (Grp=1, Msg-ID=10, S-MAC-ID= 0) D-MAC= 7 SVC=0xe Req Obj=0x0 Inst=0x0 data: 0x1
100.000000 ID= 640: (Grp=1, Msg-ID=10, S-MAC-ID= 0) D-MAC= 7 SVC=0xe Req Obj=0x0 Inst=0x0 data: 0x1
100.000000 ID= 640: (Grp=1, Msg-ID=10, S-MAC-ID= 0) D-MAC= 7 SVC=0xe Req Obj=0x0 Inst=0x0 data: 0x0
100.000000 ID=1920: (Grp=4,Msg-ID= 0, Reserved Group 4 Message)

Figure 25, DeviceNet-Protocol

9. J1939-Plugin

The J1939 plugin shows the interpretation of CAN messages in a separate window according to the J1939 protocol.

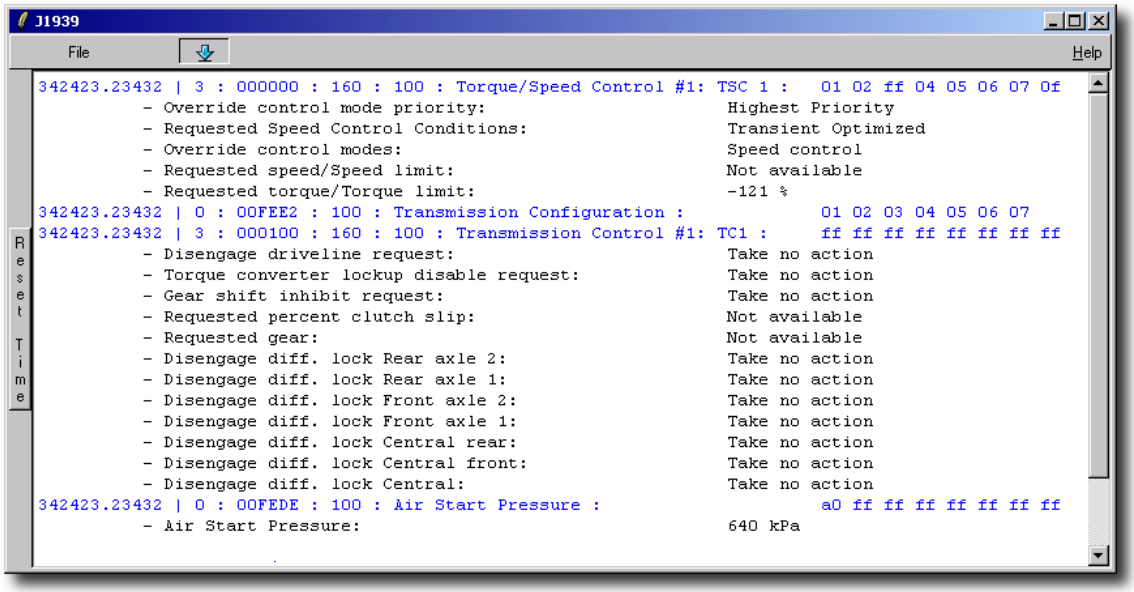

J1939
File
Help
Reset Time
342423.23432 | 3 : 000000 : 160 : 100 : Torque/Speed Control #1: TSC 1 : 01 02 ff 04 05 06 07 0f
- Override control mode priority: Highest Priority
- Requested Speed Control Conditions: Transient Optimized
- Override control modes: Speed control
- Requested speed/Speed limit: Not available
- Requested torque/Torque limit: -121 %
342423.23432 | 0 : 00FEE2 : 100 : Transmission Configuration : 01 02 03 04 05 06 07
342423.23432 | 3 : 000100 : 160 : 100 : Transmission Control #1: TC1 : ff ff ff ff ff ff ff
- Disengage driveline request: Take no action
- Torque converter lockup disable request: Take no action
- Gear shift inhibit request: Take no action
- Requested percent clutch slip: Not available
- Requested gear: Not available
- Disengage diff. lock Rear axle 2: Take no action
- Disengage diff. lock Rear axle 1: Take no action
- Disengage diff. lock Front axle 2: Take no action
- Disengage diff. lock Front axle 1: Take no action
- Disengage diff. lock Central rear: Take no action
- Disengage diff. lock Central front: Take no action
- Disengage diff. lock Central: Take no action
342423.23432 | 0 : 00FEDE : 100 : Air Start Pressure : a0 ff ff ff ff ff ff
- Air Start Pressure: 640 kPa

Figure 26, J1939 Filter settings

The filter settings allow to select source addresses that are displayed.

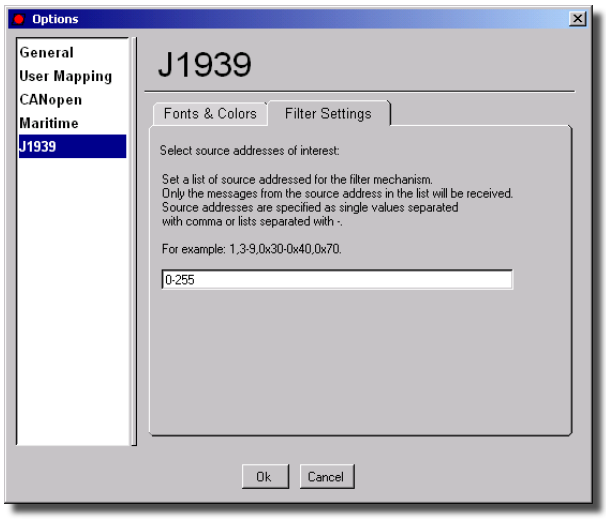


Figure 27, J1939 Filter settings

The plugin provides an API to send J1939 messages. See paragraph Extending *CAN-REport*.

The J1939 interpretation can be adopted or extended to suite application specific details. The adoption is carried out in the file *j1939def.tcl*. Two Tcl lists contains the information on how to interpret the CAN messages.

J1939MAIN	defines the text for the parameter group
J1939SUB	defines the bits and bytes of a parameter group

Example

```
 ...
#             bitPos    type Name    typeSpecParam
    59904 {     8         v   "Fuel"    {0,1,l}
 ...
```

The bit position of 8 means that all bits from the previous bit position (1 == start position) until that bit position (8) belongs to a value. It is assumed that the first 1 of the CAN-Data, is the first bit in the description. The order of the bits is important and must match the real order in the CAN message.

Type specifier	Description
v	The typeSpecParam consists of an offset, a scaling factor and a unit in the following order: `{offset,scaling,unit}` e.g: {-250,0.5,m} The unit is an optional parameter.
s	The typeSpecParam is always '-'.

b	Each combination of bits (2 bits = 0-3, 3 bits = 0-7, ...) has its own meaning. {value0,meaning 0,value1,meaning 1, value2,meaning 2, lastvalue,default meaning} e.g:
	`{0,Off,`
	`1,On,`
	`2,Error Indicator,`
	`3,Not available}`
	If not all values are specified the last value is used as default value.
	Hint: Use '-' for name and typeSpecParam to ignore bits.

h	hexadecimal values without scaling, offset, unit. The typeSpecParam is always '-'.

10. RS232-Plugin

The RS232 plugin allows to monitor a serial interface and log the data into a separate window or into the main window. The main purpose is for developers who want to see debug output from their device in relation to the CAN traffic. Furthermore the plugin provides a Tcl/Tk scripting API to send data to and recieve data from the serial interface (see chapter Programming with CAN-REport).

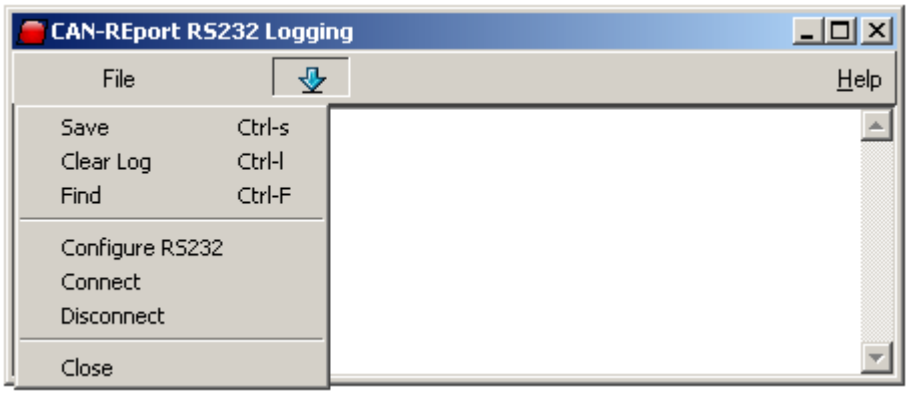


Figure 28, Logging window of the RS232 Plugin

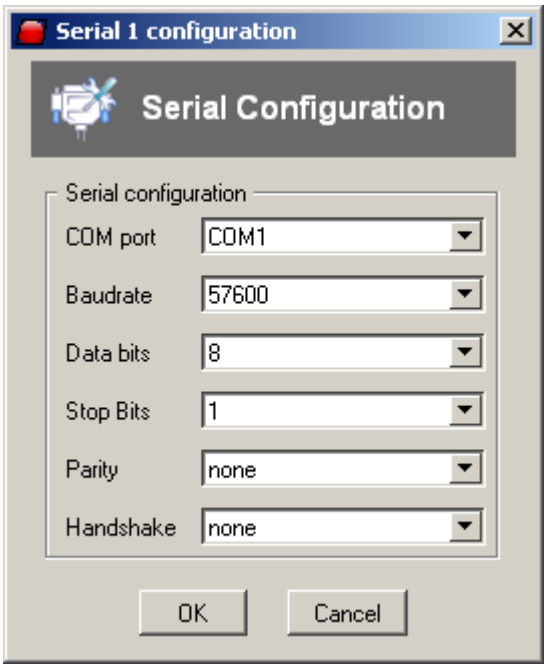


Figure 29, Configuration dialog for the serial interface

11. Trigger

With the trigger functionality *CAN-REport* is able to wait for certain CAN messages and continue recording after the desired message was received.

Trigger Event	-	Specifies the CAN message with its contents to wait for. With the help of the CANopen Message Builder it is easy to create CAN messages which conform to the CANopen protocol.
Action at trigger event	-	Select what to do when the specified CAN message was received. CAN messages can be displayed in the main window or save in a logfile.
Logfile options	-	Determine here what *CAN-REport* has to do additionally when it writes to a logfile and specify the CAN message id's for.

Table 1, Sections of the trigger dialog

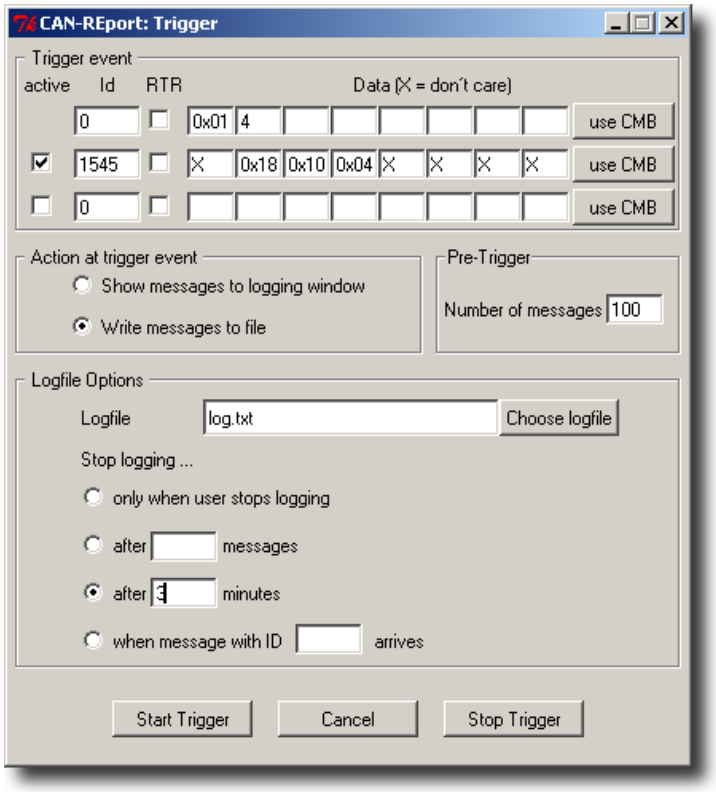


Figure 30, Trigger dialog

The process of triggering is started with the button "Start Trigger". During the trigger process the dialog stays on top of the application and recording of messages in the main window stops. After the specified CAN message was received the trigger dialog disappears.

11.1. Defining a trigger CAN message

Up to three different CAN message can be specified as trigger events. The second and third CAN message are activated through the checkbutton labeled "active". The first CAN message is active by default. The input in the fields CAN-ID and data can be specified as decimal, hexadecimal or octal numbers. Hexadecimal numbers are prepended with "0x" and octal numbers with "0". A blank input field determines the end of the CAN message. Any data byte after a blank input field is ignored. The length is then calculated by the number of non empty input fields. In order to ignore the contents of a byte mark it as don't care by entering an "X" into an edit field instead of a numerical value. Then its value is not compared with the received message.

Examples:

Figure 31, Trigger example 1

CAN message with 8 bytes length. Byte 4 to 7 are marked as don't care. Numbers used in decimal hexadecimal and octal

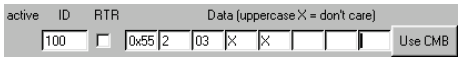


Figure 32, Trigger example 2

CAN message with 5 bytes length. Byte 4 and 5 are marked as don't care. Numbers used in decimal hexadecimal and octal

Figure 33, Trigger example 3

Only the CAN-id is compared. The content and length of the CAN message is not evaluated.

11.2. Defining a trigger Remote Request CAN message

Defining a Remote Request CAN message follows the same rules as a "normal" CAN messages. Setting the expected length of the RTR message is done with the don't care bytes. Example:

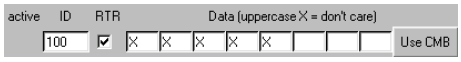


Figure 34, Trigger example 4

The *CAN-REport* triggers on a RTR message with CAN id 0x64 and a length of 5 bytes.

11.3. CANopen Message Builder

The CANopen Message Builder eases creating of CAN messages for the trigger which conform to the CANopen protocol. It provides input masks for the CANopen services NMT, PDO, SDO, HEARTBEAT and EMCY. Figure 35 shows the input mask for the NMT service. The CANopen Message Builder is only available with the CANopen

Plugin.

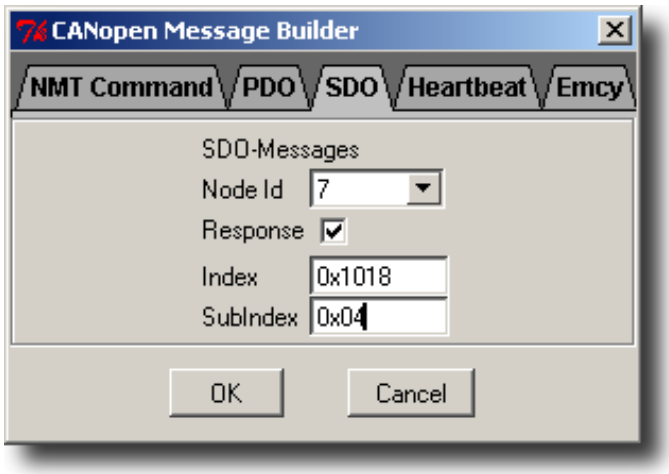


Figure 35, Input dialog for NMT service of the NMT service

12. Filter

12.1. Hardware Filter

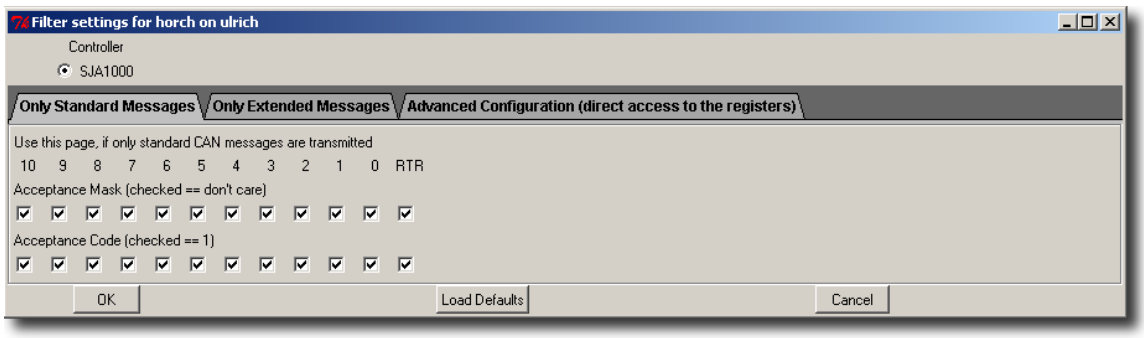

Filter settings for horch on ulrich
Controller
SJA1000
Only Standard Messages
Only Extended Messages
Advanced Configuration (direct access to the registers)
Use this page, if only standard CAN messages are transmitted
10 9 8 7 6 5 4 3 2 1 0 RTR
Acceptance Mask (checked == don't care)
Acceptance Code (checked == 1)
OK
Load Defaults
Cancel

Figure 36, Filter dialog

The filter dialog provides access to the register of the acceptance mask of the CAN controller SJA1000. The usage of acceptance filtering in the CAN-Controller reduces the interrupt load. There are three ways of defining the acceptance mask:

1. Input for the acceptance mask for CAN 2.0 A starting with the MSB on the left side.
2. Input of the acceptance mask for CAN 2.0 B starting with the MSB on the left side.
3. Input of the acceptance mask as it is defined in the register of the CAN controller.

This feature depends on the hardware and driver respectively you are using. At the moment the horch server for *can4linux* and *CPC (SJA1000)* support setting the acceptance mask registers. The other supported hardware interfaces (see Restrictions) don't support this.

Attention: Although it is possible to set the two least significant bits of the CAN identifier the SJA1000 doesn't use these two bit during acceptance filtering.

Examples:

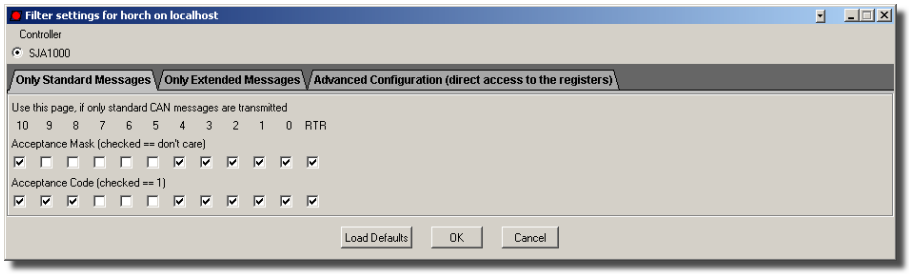

Filter settings for horch on localhost
Controller
SJA1000
Only Standard Messages
Only Extended Messages
Advanced Configuration (direct access to the registers)
Use this page, if only standard CAN messages are transmitted
10 9 8 7 6 5 4 3 2 1 0 RTR
Acceptance Mask (checked == don't care)
Acceptance Code (checked == 1)
Load Defaults
OK
Cancel

Figure 37, Acceptance register values for NMT filtering. CAN identifiers range 0x700 - 0x71f, ie. CANopen nodes 1 to 31

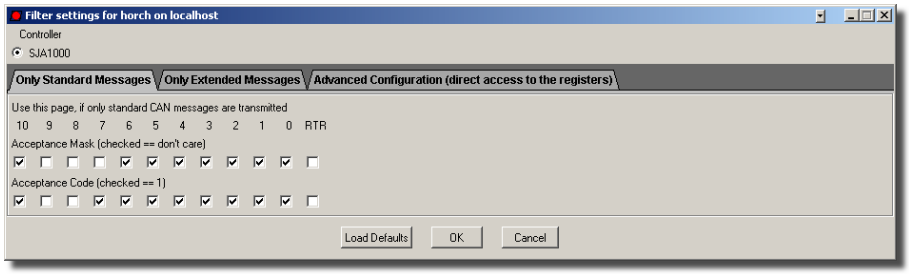

Filter settings for horch on localhost
Controller
SJA1000
Only Standard Messages
Only Extended Messages
Advanced Configuration (direct access to the registers)
Use this page, if only standard CAN messages are transmitted
10 9 8 7 6 5 4 3 2 1 0 RTR
Acceptance Mask (checked == don't care)
Acceptance Code (checked == 1)
Load Defaults
OK
Cancel

Figure 38, Acceptance register values for EMCY filtering. CAN identifiers range 0x80 - 0xff, ie. CANopen nodes 1 to 127

12.2. Software Filter

The software filter works with all drivers that make use of the horch server. With it only the defined message ids are received. The message ids that are to be received are given in a list of values. Each values is separated by comma from each other. Ranges of message ids are given in format: *<lowValue>'-'<highValue>*

Software Filter
Software Filter
Software CAN filter
Set a list of message ids for the software filter mechanism.
Only the message ids in the list will be received.
CAN message ids are specified as single values separated with comma or lists separated with -.
For example: 1,3-9,0x580-0x600,0x701.
0x0-0x1fffffff
OK
Cancel

Figure 39, Filter dialog

13. Programming with *CAN-REport*

13.1. Extending *CAN-REport*

With the help of the integrated console convenient commands for the interactive access on the CAN network are available and not only for device developers. These commands can be used in test scripts. Commands like **`wr`** for the sending of messages or **`wait`** for the synchronization with CAN messages on the network belong to this purpose.

Overview of user functions

can1.wait	-	*Wait for CAN message with id or until timeout*
can1.wait2	-	*Wait for CAN message with id and type or until timeout*
can1.wr	-	*Send CAN message with standard identifier*
can1.wrx	-	*Send CAN message with extended identifier*

Detailed function description

can1.wait `id timeout`

Wait for CAN message with id or until timeout

This function is deprecated please use wait2

Arguments

`id`	- can be an integer, or hex (0x...) or "any"
`timeout`	- in ms, defaults to 10000 == 10s

Results

String with message
String: "wait timeout"

can1.wait2 `id type timeout`

Wait for CAN message with id and type or until timeout

Arguments

`id`	- can be an integer, or hex (0x...) or "any"
`type`	- any combination of s,x,D,R
	- s standard; x extended D data; R- RTR
`timeout`	- in ms, defaults to 10000 == 10s

Results

String with message
String: "wait timeout"

Example

```
wait2 12 R   ; wait 12s for RTR ID of any format s or x
wait2 10 sR  ; wait 10s for standard RTR ID
wait2 any    ; wait for the next message
```

can1.wr

Send CAN message with standard identifier

Arguments

`args`	- a CAN message. It is specified in the format
	- [r] id data-bytes
	- the optional ’r’ denotes the message as RTR CAN message

Results

- nothing

Example

```
wr r 0x620 0x30 0xff
wr 0x20    0x69 0x55
```

can1.wrx

Send CAN message with extended identifier

Arguments

`args` - a CAN message. It is specified in the format
- [r] id data-bytes
- the optional 'r' denotes the message as RTR CAN message

Results

nothing

Example

```
wrx r 0x820 0x30 0xff
wrx 0x20    0x69 0x55
```

The following figure shows an example session at the **console** entering a command sequence:

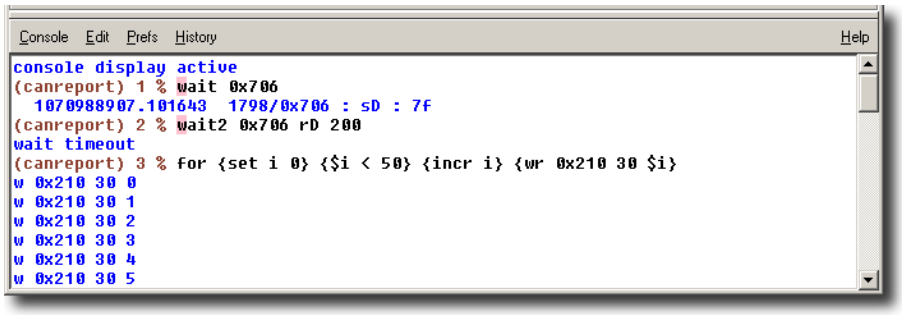

Console Edit Prefs History Help
console display active
(canreport) 1 % wait 0x706
1070988907.101643 1798/0x706 : sD : 7f
(canreport) 2 % wait2 0x706 rD 200
wait timeout
(canreport) 3 % for {set i 0} {$i < 50} {incr i} {wr 0x210 30 $i}
w 0x210 30 0
w 0x210 30 1
w 0x210 30 2
w 0x210 30 3
w 0x210 30 4
w 0x210 30 5

Figure 40, Entering interactive commands

The resulting messages at the CAN network are shown in the next figure:

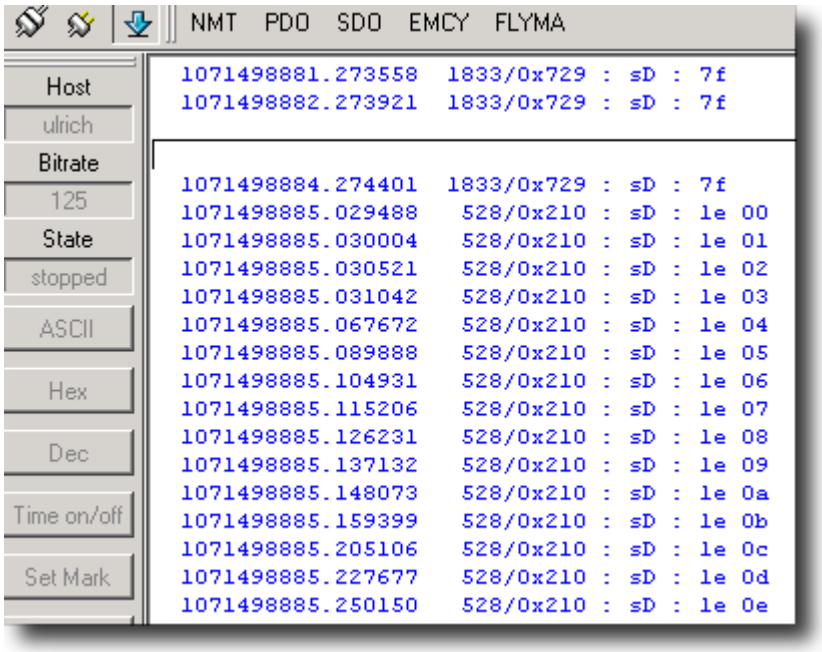


Figure 41, Results of the commands of figure 40

The commands can be combined into sequences or procedures. All semantics of modern high level languages are available.

Particularly for commissioning and error analysis a highly precise time resolution of the received CAN messages on the network is necessary. The time represented by *CAN-REport* is influenced in this case only from the used hardware. A time resolution up to µs can be achieved by using CAN interface boards under the LINUX operating system.

The available recording functions allow to store the results of an entire test run, but also the contents of logging windows, in separate files. As well it is possible to store the CAN messages directly in a file without displaying them on the screen. These files can be interpreted later by the CANopen or user defined plugins. This is especially practical at high bus load.

Complex scripts can be aborted with the stop button from the toolbar. To make this work the global variable `ST(userStop)` has to be checked. As soon as the stop button is pressed the value `stop` is assigned to the global variable `ST(userStop)`.

```
proc yourTest { } {

    if { "stop" eq $::ST(userStop) } {
     # do whatever is necessary to stop

     set ::ST(userStop) ""
     return
    }
}
```

13.2. Interpretation of CAN messages

The *CAN-REport* features the interpretation of CAN-messages. For the application layer protocol CANopen this was already done. When the button "User plugin" in the file menu is pressed a file called <application.tcl> is loaded and a button is placed on the toolbar. This file has to be placed in the working directory. When the button in the toolbar is pressed a window pops up. The user can write an plugin for the *CAN-REport* and display his own interpretation of the CAN messages in this window. The widget path to the user window is `.usr`. For writing your own CAN interpretation the template file application.tcl is provided. It contains the procedure `application`. This procedure is called on each reception of a CAN message.

```
proc application { msg } {

}
```

This procedure takes the parameter `msg`. It contains the complete message with timestamp, id, data type and the data. Each field is separated by spaces.

```
        11.123  256/0x100 : sD : 00 01 02 03 04 05 06 07
#       ^       ^           ^    ^
#         time  id dec/hex type data 1..8 (if available)
```

There are already functions for extracting the data.

Overview of user functions

InsertText	-	*Insert text <string> into user window*
can1.on	-	*Register script or procedure for a message*
get_can_id	-	*extract CAN message identifier from message string*
get_data	-	*return the data of a the message*
get_dlc	-	*return the data length code of the message*
get_time	-	*return time of message if time was set*
get_type	-	*return the type of the message*
get_type2	-	*return the type of the message*
is_std	-	*check if the message is a standard CAN message*

Detailed function description

InsertText `win string tag`

Insert text <string> into user window

Arguments

`win`	- widget-path .usr
`string`	- Text to insert into the text-widget
`tag`	- tag, that defines the font the tags stdout and stderr are already defined

Results

Inserts a text in the user window

Example

```
InsertText .usr "hello world!" stdout
InsertText .usr "$msg" stdout
```

can1.on `type`

Register script or procedure for a message that will by exectued when the message is received. The message is provide in the Tcl array this. The array has the elements: time, id, type, dlc, data. The element dlc is the data length code of the CAN message.

Arguments

type
Id
script

Results

`0` - success
`1` - Error; the error messgae is printed to the console

Example

```
on message 0x12 { wr 0x32 0 1 2 }
on message 0x11 {
      puts $this(id)
}
```

get_can_id `msg`

extract CAN message identifier from message string

Arguments

msg the standard message string provided for the procedure application {}

Results

the CAN-Id as decimal value

get_data `msg`

return the data of a the message

Arguments

msg the standard message string provided for the procedure application {}

Results

list containing the databyes of the CAN message
in case it is an RTR message an empty string is returned:

get_dlc `msg`

return the data length code of the message

Arguments

msg the standard message string provided for the procedure application {}

Results

an integer value between 0 and 8 for a valid msg
an empty string for an invalid msg

get_time `msg`

return time of message if time was set

Return time information if time was set. When time wasn't set the an empty string is returned

Arguments

msg the standard message string provided for the procedure application {}

Results

`time`	- when time is on
`empty string`	- when time is off

get_type `msg`

return the type of the message

Arguments

msg the standard message string provided for the procedure application {}

Results

D	- Data frame
R	- Remote frame

get_type2 `msg`
return the type of the message

Arguments

msg the standard message string provided for the procedure application {}

Results

`sD` - base Data frame
`sR` - base Remote frame
`xD` - extended Data frame
`xR` - extended Remote frame

is_std `msg`
check if the message is a standard CAN message

Arguments

msg the standard message string provided for the procedure application {}

Results

`1` - if the message is a standard CAN-Message
`0` - if the message is not a standard CAN-Message

13.2.1. Example

```
#The procedure application receives the message and interprets it.
#It’s an example for an user-defined interpretation of CAN-Messages.
#
proc application { msg } {

    set tag stdout
    set msg ""

    # get id
    set id [get_can_id $msg]
    if { $id eq "" } {
     return
    }

    if { $id == 0x201 } {
     # extract data
     set data [get_data $msg]

     set len [get_dlc $msg]
     for { set i 0 } { $i < $len } { incr i } {
         set byte$i [lindex $data $i]
     }
     if { $byte0 == 0x55 } {
         set msg "Start processing"
     }
    }

    if { $id == 0x281 } {
     # extract data
     set data [get_data $msg]

     set len [get_dlc $msg]
     for { set i 0 } { $i < $len } { incr i } {
         set byte$i [lindex $data $i]
     }
     if { $byte0 == 0x1F } {
         set msg "zero limit switch reached"
     }
    }

    InsertText .usr $msg $tag
    return
}
```

13.3. Multichannel functions

The multichannel plugin enhances the following commands to be used on each CAN line. The CAN line is selected by prepending the word: '`can1.`' and '`can2.`'. So the resulting command is for example `can1.wr` and `can2.wr`.

Standard commands

wr
wrx
wait
wait2
on

CANopen commands

canopen.sdo.w
canopen.sdo.r
canopen.sdo.setSdoTimeout
canopen.sdo.getSdoTimeout
canopen.sdo.setRemoteID
canopen.sdo.getRemoteID
canopen.nmt.start
canopen.nmt.stop
canopen.nmt.preop
canopen.nmt.resetAppl
canopen.nmt.resetComm

Table 2, Commands enhanced by the Multi channel plugin

13.4. Testscript commands

These commands are helpful for creating test protocolls.

13.5. CANopen functions

The CANopen-Plugin provides some basic functionality for accessing the object directory of a CANopen SDO-Server. Therefore the *CAN-REport* acts like an SDO-Client. The read and write function use Expedited-SDO-Transfer. With Expedited-SDO-Transfer a maximum of 4 Bytes can be transmitted. The datatypes are specified as in the gateway standard DS309-3.

Datatype	Meaning
u8	8bit unsigned char
i8	8bit signed char
u16	16bit unsigned char
i16	16bit signed char
u32	32bit unsigned char
i32	32bit signed char

Table 3, CANopen datatypes

Important note
The commands listed below are the new commands. For backward compatibility the old commands without '`can1.canopen.sdo`' are still available.

Overview of CANopen user functions

can1.canopen.nmt.preop	-	*send NMT command. Set specified node to Pre-Operational.*
can1.canopen.nmt.resetAppl	-	*send NMT command. Reset application of the specified node.*
can1.canopen.nmt.resetComm	-	*send NMT command. Reset communication of the specified node.*
can1.canopen.nmt.start	-	*send NMT command. Set all nodes to Operational*
can1.canopen.nmt.stop	-	*send NMT command. Stop application of the specified node.*
can1.canopen.sdo.getRemoteID	-	*get remote Id that is used for SDO transfers*
can1.canopen.sdo.getSdoTimeout	-	*get timeout for SDO-Transfer*
can1.canopen.sdo.r	-	*read data with expetited SDO transer*
can1.canopen.sdo.setRemoteID	-	*set remote Id for SDO transfers*
can1.canopen.sdo.setSdoTimeout	-	*set timeout for SDO-Transfer*
can1.canopen.sdo.w	-	*write data with expetited SDO transer*

Detailed function description

can1.canopen.nmt.preop `nodeId`

send NMT command. Set specified node to Pre-Operational.

This function is only available with the CANopen extension.

Arguments

`id` - node id, default is 0 (all)

Results

nothing

can1.canopen.nmt.resetAppl `nodeId`

send NMT command. Reset application of the specified node.

This function is only available with the CANopen extension.

Arguments

`id` - node id, default is 0 (all)

Results

nothing

can1.canopen.nmt.resetComm `nodeId`

send NMT command. Reset communication of the specified node.

This function is only available with the CANopen extension.

Arguments

`id` - node id, default is 0 (all)

Results

nothing

can1.canopen.nmt.start `nodeId`

send NMT command. Set all nodes to Operational

This function is only available with the CANopen extension.

Arguments

`id` - node id, default is 0 (all)

Results

nothing

can1.canopen.nmt.stop `nodeId`

send NMT command. Stop application of the specified node. The device will not respond to SDO commands.

This function is only available with the CANopen extension.

Arguments

`id` - node id, default is 0 (all)

Results

nothing

can1.canopen.sdo.getRemoteID

get remote Id that is used for SDO transfers

This function is only available with the CANopen extension.

Arguments

Results

return remote ID

can1.canopen.sdo.getSdoTimeout
get timeout for SDO-Transfer

This function is only available with the CANopen extension.

Arguments

Results

return SDO timeout

can1.canopen.sdo.r `idx sidx dt`
read data with expetited SDO transer

This function is only available with the CANopen extension. The node id has to be set previously

Arguments

`idx` - Index of the server object
`sidx` - SubIndex of the server object
`dt` - Datatype of the value

Results

`"ERROR ..."` - Error, reason is given
`value` - Success

can1.canopen.sdo.setRemoteID `nid`
set remote Id for SDO transfers

This function is only available with the CANopen extension. The initial value is 0x20.

Arguments

`nid` - new node id

Results

`0` - Error
`1` - Success

can1.canopen.sdo.setSdoTimeout `val`
set timeout for SDO-Transfer

This function is only available with the CANopen extension. The initial value is 1000 ms.

Arguments

`val` - new timeout (ms)

Results

can1.canopen.sdo.w `idx sidx dt val`
write data with expetited SDO transer

This function is only available with the CANopen extension. The node id has to be set previously

Arguments

`idx` - Index of the server object
`sidx` - SubIndex of the server object
`dt` - Datatype of the value
`val` - value itself

Results

`"ERROR ..."` - Error, reason is given
`"OK"` - Success

13.6. DeviceNet functions

Overview of DeviceNet user functions

devicenet::allocate	-	*allocate Predefined Master/Slave Connection Set*
devicenet::closeExplConnection	-	*close Explicit Messaging Connection Request (UCMM)*
devicenet::getAttributeSingle	-	*read an attribute*
devicenet::getDMAC	-	*get destination MAC-Id*
devicenet::getSMAC	-	*get source MAC-Id*
devicenet::openExplConnection	-	*open Explicit Messaging Connection Request (UCMM)*
devicenet::poll	-	*poll a device*
devicenet::setAttributeSingle	-	*set a single attribute with an explict connection*
devicenet::setDMAC	-	*set destination MAC-Id.*
devicenet::setSMAC	-	*set source MAC-Id*
devicenet::ucmmGetAttributeSingle	-	*get Single Attribute using the UCMM service*
devicenet::ucmmReadManId	-	*read manufacturer ID using UCMM service*
devicenet::ucmmReadSerial	-	*read serial number using UCMM service*
devicenet::ucmmSetAttributeSingle	-	*execute SetAttributeSingle service by UCMM*

Detailed function description

devicenet::allocate

allocate Predefined Master/Slave Connection Set

Referring to Chapter 5, (page 5-57). Source MAC_ID and Allocator's MAC_ID are the same, that means the requester is the user of the connection After establishing a connection, the following services are available: - Polled I/O master poll request : data fro master to slave slave poll response : data from slave to master - Bit-Strobe I/O Connection - Change of State/Cyclic I/O Connection

Arguments

`args` - allocation choice(es)
which connections are needed, is one:
explicit | polled | strobe | cos | cyclic | ack
or a mix of these
may be polled or cos without explicit isn't possible (1791D8B8P)

Results

`format` - message body format used by the device (as returned from device)

devicenet::closeExplConnection `dmac_id ciid`

close Explicit Messaging Connection Request (UCMM)

Close Explicit Messaging Connection Request (UCMM) to node.

Arguments

`dmac_id` - destination MAC-ID
`ciid` - connection instance id, got from Open Explicit Messaging Req.

Results

`Ok` - success
`"Error timeout"` - failure

devicenet::getAttributeSingle `class instance attribute`
read an attribute

Read an object/instance Attribute using predefined Master/Slave Connection Set

Arguments

`class` - class of desired attribute
`instance` - instance of desired attribute
`attribute` - attribute id of desired attribute

Results

Returns the value of an attribute addressed by class, instance and attribute in the case the attribute contains up to four byte, e.g. returns a not fragmented response, the result is an integer value If the node returns a fragmented response result is a list of byte values. Each value is represented as integer value between 0 and 255.

devicenet::getDMAC
get destination MAC-Id

Get destination MAC-Id

Arguments

`nothing` -

Results

`DMAC_ID` - destination MAC-Id

devicenet::getSMAC
get source MAC-Id

Get source MAC-Id.

Arguments

`nothing` -

Results

`SMAC_ID` - source MAC-Id

devicenet::openExplConnection `dmac_id`
open Explicit Messaging Connection Request (UCMM)

Open Explicit Messaging Connection Request (UCMM) to node. If no destination MAC-ID is given then it defaults to the MAC-ID set with setDMAC.

Protocol: Byte 0 - dest mac id; Byte 1 - request service code - 0x4b; Byte 2 - message body format: 0 means 8bit for class. 8bit for instance; Byte 3 - source (message group << 4) + message id message group 0 or 3 ==> the client is requesting a fixed connection in Message Group_1

Arguments

`dmac_id` - (optional) destination MAC-ID

Results

`instance Id` - Success
`"Error ..."` - failure

devicenet::poll `dmac_id`
poll a device

Used to set data using the predefined polled I/O connection.

Restriction: no fragmented messages yet

Arguments

`dmac_id`	- destination mac id of polled device
`args`	- (optional) bytes to be send to the device if no data are given, the "poll idle" message is send to the destination mac id

Results

nothing

devicenet::setAttributeSingle `class instance attribute type data`
set a single attribute with an explict connection

Use the explicit connection to set the attribute of an instance of an object If the data type is <= 3 bytes a non-fragmented transfer can be used, otherwise only fragmented is possible

Arguments

`class`	- class of desired attribute
`instance`	- instance of desired attribute
`attribute`	- attribute id of desired attribute
`type`	- data type of the value to write to the attribute one of u8, u16, u32
`value`	- value to write

Results

`timeout`	- timeout
`error message`	- error response received
`OK`	- success

devicenet::setDMAC `id`

set destination MAC-Id.

Set destination MAC-Id. Used to address the destination device.

Arguments

id	- destination MAC-Id

Results

`nothing` -

devicenet::setSMAC `id`

set source MAC-Id

Set source MAC-Id

Arguments

`id` - source MAC-Id

Results

`nothing` -

devicenet::ucmmGetAttributeSingle `dmac_id class instance attribute`
get Single Attribute using the UCMM service

Uses SVC=0xe to request the attribute of the given instance of a class data type can be any type between 1 and 4 bytes. As source-id the global value of variable SMAC_ID is used

For segmented message one has to look if the response is segmented using a loop while segmented ist TRUE, with a time out condition

Arguments

`dmac_id` - destination MAC
`class` - Class of desired attribute
`instance` - Instance of desired attribute
`attribute` - Attrbute id of desired attribute

Results

`values` - list of byte values
`"Error: ..."` - describing error condition

devicenet::ucmmReadManId `dmac_id`
read manufacturer ID using UCMM service

Read manufacturer ID using a new connection got from UCMM the connection is closed after reading. If no destination MAC-ID is given then it defaults to the MAC-ID set with setDMAC.

Arguments

`dmac_id` - (optional) destination MAC ID

Results

`manufacturer ID` -

devicenet::ucmmReadSerial `dmac_id`

read serial number using UCMM service

Read serial number using a new connection got from UCMM The connection is closed after reading. If no destination MAC-ID is given then it defaults to the MAC-ID set with setDMAC.

Arguments

`dmac_id` - (optional) destination MAC Id

Results

`serial number` -

devicenet::ucmmSetAttributeSingle `dmac_id class instance attribute type data`

execute SetAttributeSingle service by UCMM

Execute SetAttributeSingle service by UCMM

Arguments

`dmac_id` - destination MAC-Id
`class` - class of desired attribute
`instance` - instance of desired attribute
`attribute` - attribute id of desired attribute
`type` - data type
`data` - data value

Results

OK success
error text

13.7. J1939 functions

Overview of J1939 user functions

::j1939::sendPGN - *Send a parameter group as a CAN message*

Detailed function description

::j1939::sendPGN `pgn prio src dest`

Send a parameter group as a CAN message

This function is only available with the J1939 extension.

Arguments

`pgn` - Parameter Group Number (decimal)
`prio` - Priority (0-7)
`src` - Source address
`dest` - Destination address (opt.) {}
`args` - Real values of the parameters (e.g. 100 für 100 kPa, use 0-3 for bitwise values)

Results

`""` - nothing

Restrictions
The data page is assumed to be 0.

Example

```
::j1939::sendPGN 65257 6 32 {} 500.5 1000.5
```

13.8. RS232 functions

Overview of RS232 user functions

serial1.close	-	*Close the serial/RS232 interface.*
serial1.open	-	*Open the serial/RS232 interface.*
serial1.read	-	*Read data from the serial interface.*
serial1.write	-	*Send data via the serial interface.*

Detailed function description

serial1.close

Close the serial/RS232 interface.

Arguments

none -

Results

`0`	- on success
`error string`	- on failure

serial1.open

Open the serial/RS232 interface.

Arguments

none -

Results

`0`	- on success
`error string`	- on failure

serial1.read

Read data from the serial interface.

Arguments

`-timeout <val>` - timeout for reading, default is 0 i.e. wait forever

Results

`0` - on failure
`>0` - on success, it is the numbe of bytes written

Example

```
serial1.read
serial1.read -tiemout 1000
```

serial1.write

Send data via the serial interface.

Arguments

`data` - a single string

Results

`0` - on failure
`>0` - on success, it is the numbe of bytes written

13.9. Extending the user interface

With the script language Tcl/Tk also elements of the graphic interface can be accessed and changed. With that the representation of values or test outputs is expandable in user specific ways. It is possible to use provided or own graphical objects to create test and control applications. With the help of the multifaceted possibilities for displaying and reporting data test sequences can be carried out automatically and over the network.

port GmbH can perform special adoptions to the GUI or behavior to meet the CAN or CANopen device characteristics for service or assembly testing.

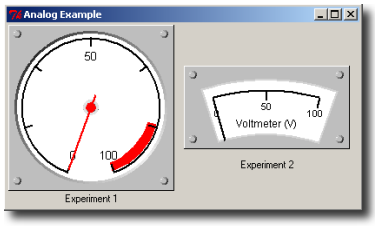


Figure 42, Example Analog

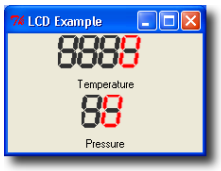


Figure 43, Example LCD

The examples make usage of the User Plugin mechanism of *CAN-REport*. Therefore the contents of an example has to be copied to the directory of the executable *CAN-REport* before it can be used.
However, when the file application.tcl of an example is loaded with File→Open the user interface is shown but not updated by incoming CAN messages.

Demo-Mode

In demo mode the console plays a demo script and opens the provided examples.

14. Configuration

14.1. Serial interfaces

The particular hardware can be selected in the following configuration dialog. *Serial Port* allows the configuration of the serial port (COMx on Windows and /dev/ttySx on Linux respectively) and the CAN baud rate can be configured by *Bitrate*.

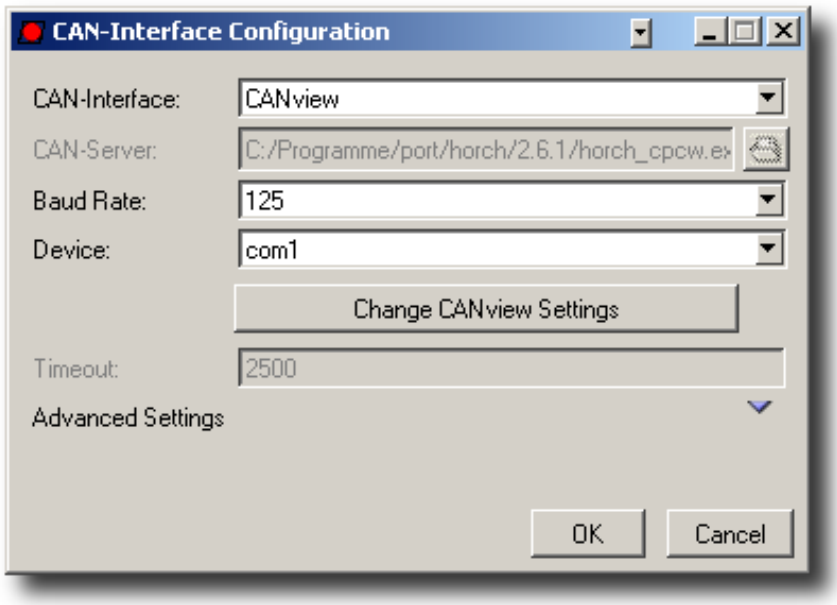


Figure 44, Configuration dialog

14.1.1. CAN232 - Configuration

Please ensure that your CAN232 is connected properly to the power supply and to the PC. After pressing the button *Configure CAN232* the *CAN-REport* tries to connect to the CAN232 by scanning the RS232 baud rates 230400, 115200, 57600, 38400, 19200, 9600, 2400. On success a dialog window is opened to change the RS232 baud rate.

For support of the CANUSB the compatibility mode is used. The device is controlled via the virtual serial COM port.

Note: Baudrate scanning does not work under Windows 98. The current used baudrate has to be inserted by hand.

14.1.2. CANview - Configuration

After pressing the button *Change CANview Settings* a dialog window is opened to configure the CANview settings.

Please regard that it does **not** configure the device. Instead the *CAN-REport* has to be configured according to the device settings. To change the device settings directly please use the *RM Device Configurator*.

15. System Requirements

CAN-REport is executables on PC's with Microsoft Windows and UNIX/LINUX systems.

Operating system:	Windows 2000,XP,Vista, LINUX
Processor:	Pentium and higher
RAM:	64 MByte
hard disk space:	10 MByte